Robotics
Lab 1: Basic Exercises

Group 3
Sebastian Grasdijk (s1972243) & Laura Baakman (s1869140)

November 23, 2015

The code for this assignment can be found in the branch `group3`, it is tagged `basics`.

Exercise 1

Note that we have used the Xtion to complete this exercise, instead of the provided bag file.

For this exercise we have defined a package called `basics`. In this package we have created a folder `launch`, in which we placed the launch file presented presented in listing 1.

Listing 1: The launch file for the camera, `camera.launch`.

```
<launch>
<node
    pkg="tf"
    type="static_transform_publisher"
    name="link1_broadcaster"
    args="-2 -0.4 1.4 0 0.71 -0.12 /base_link camera_link 100" />
</launch>
```

The tag `<launch>` is the root element of the file `camera.launch`, it indicates that we are defining a launch file. Launch files allow us to start multiple nodes at once. The child tag `<node>` , of `<launch>` indicates which ROS node we would like to launch. The attributes `pkg` and `type`, the name of the executable, attribute identify which program ROS should run to start this node. The attribute `name` assigns a name to the node.

We use a `static_transform_publisher` which publishes a static coordinate transform. This executable requires several arguments, which are passed via the attribute `args`. One should call the `static_transform_publisher` as follows

```
    static_transform_publisher x y z yaw pitch roll frame_id
   child_frame_id period_in_ms
```

The `x`, `y`, `z` indicate an offset in the x, y and z-direction. We have used a z offset of 1.400 m to correct for the height of the stand of the Xtion. The x and y offset ensure that the frame is placed in the middle of the point cloud, presented in Figure 1.

The `yaw`, `pitch` and `roll` define the rotation around respectively the z, y and x axis in radians. The pitch 0.7200 rad corrects for the angle of the Xtion, which translates to around $^1/_4\pi$. The `yaw` and `roll` should be zero in a perfectly calibrated stand, however we needed to tweak the roll

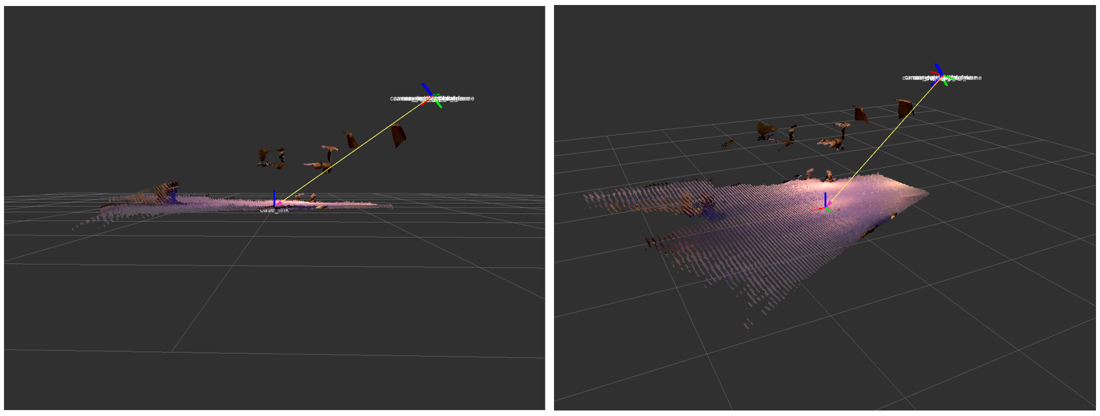
Figure 1: The point as visualized by rviz, under two different angles, after the application of the static transformation defined in listing 1.

a bit to get the point cloud were we wanted it, due to the camera not being perfectly parallel to the floor.

Following the earlier mentioned information there are two values, which are frame_id and child_frame_id. These refer to the link and its parent link. In our model they are the base link and the camera link, respectively. The camera link shows the base point from which the camera records the information it receives, and the base link works as a reference point for the floor in the case of the information that we received. This can be seen in Figure 1 . The yellow line shows the link between the base link and the camera link.

We have set period_in_ms to 100 ms, which is the value recommended by the ROS documentation.

Exercise 2

We added the file with our adjustments to the file alice_description/complete_model.urdf.xarco, see listing 2.

Listing 2: Adding the file assignment.urdf.xacro.xml to complete_model.urdf.xacro.xml

```
<!-- Assignment addition -->
<xacro:include filename="$(find alice_description)/urdf/assignment.
   urdf.xacro" />
```

Listing 3 presents the link associated with the bar, top_middle_bar that is placed on top of the middle_bar. We have assumed that other than its length the bar should have the same dimensions as the middle_bar, consequently its mass is ${}^{0.3}/_{0.68}$ of the mass of the middle_bar. The inertia of this bar is the same as that of the middle_bar. The scale property of the mesh element indicates the size the size of the bar.

Listing 3: The link of the bar placed on top of the middle_bar.

```
<link name="top_middle_bar">
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <inertial>
        <mass value="4.41176"/>
        <origin xyz="0 0 0"/>
        <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <visual name="visual">
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://alice_description/models/box.dae
" scale="0.02 0.04 0.3" />
        </geometry>
    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://alice_description/models/box.dae
" scale="0.02 0.04 0.3" />
        </geometry>
    </collision>
</link>
```

The top_middle_bar is connected with the middle_bar with the joint middle_bar_to_top_middle_bar. This joint is fixed, since the these two links cannot move relative to each other. We have given the joint an offset of ${}^{0.68}/_{2} + {}^{0.3}/_{2} + 0.02$. This ensures that the top_middle_bar is placed on top of the middle_bar and is exactly 30 cm above the top of the middle_bar.

Listing 4: The definition of the joint of the bar in assignment.urdf.xacro.xml.

```
<joint name="middle_bar_to_top_middle_bar" type="fixed"> <!--
   Connection between Middle bar and bottom rectangular box -->
    <origin rpy="0 0 0" xyz="0 0 ${0.68 / 2 + 0.3 / 2 + 0.02}"/>
    <child link="top_middle_bar"/>
    <parent link="middle_bar"/>
</joint>
```

Listing 5: The definition of the joint middle_xtion_top_middle_bar in assignment.urdf.xacro.xml.

```
<joint name="middle_xtion_top_middle_bar" type="fixed"> <!--
   Connection between top bar and front -->
    <origin rpy="0 ${M_PI-M_PI/4} 0" xyz="0.0 0.0 ${0.15+0.0222}"/>
    <child link="middle_xtion_link"/>
    <parent link="top_middle_bar"/>
</joint>
```

The complete code can be found in Appendix A. Listing 5 presents the definition of the joint middle_xtion_top_middle_bar. The child link middle_xtion_link is not shown here because it is a copy from the front_xtion_link. In order to get the correct angle, we tilted the camera $3/4 * \pi$ so that it was facing downwards.

Because the camera had to be on top of the bar, we calculated the distance for the joint, that it had to be upwards on the z-axis. Using basic geometry we calculated that the camera would have to be placed $15 + 2.121$ upwards on the z-axis in order to put it on the top of the bar, with the lowest point still touching the bar.

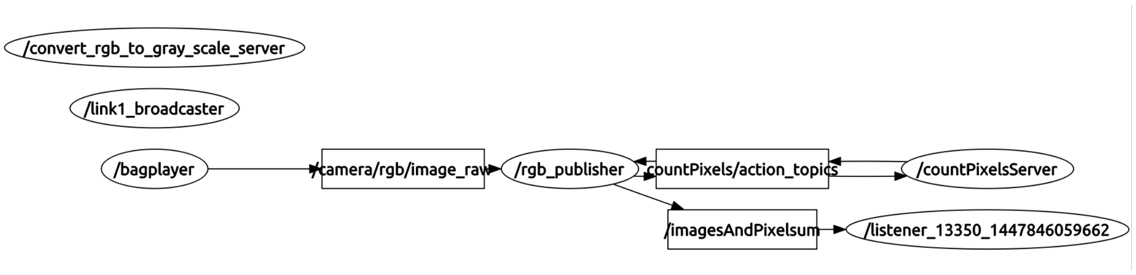

Figure 2: The connections between the different nodes.

Listing 6: The RGBtoGrayScale service.

```
sensor_msgs/Image rgb
---
sensor_msgs/Image grayscale
```

Exercise 3

The general structure of the code written for this exercise is shown in Figure 2. Note that although the node convert_rgb_to_gray_scale_server is unconnected in this image, it has a link with rgb_publisher. Since it simply is a service and not an actual node publishing information, it shows as unconnected when a graph is generated. Below we discuss the route of an RGB image that is played from the bag file to the point where it is displayed in RGB and gray scale on the screen and the number of pixels in both those images have been counted.

In lieu of a camera the images are published from a bag, this is done by the node bagplayer. This nodes publishes RGB images on the topic /camera/rgb/image_raw. The rgb_publisher subscribes to this topic, see the method subscriber_callback in listing 11 in Appendix B. In the callback of this function, subscriber_callback the received RGB image is send to the convert_rgb_to_gray_scale_server. Since we only need to send the RGB image to this server we use the message type sensor_msgs.msg.Image.

The convert_rgb_to_gray_scale_server receives an RGB image and returns an grayscale image, both of type sensor_msgs/Image, see listing 6. The service converts the image to a CV image. The CV image is converted to a gray scale image which is converted back to the ROS image format, see listing 12. This image is send back to the requesting node as a sensor_msgs .msg.Image. We do not add the RGB image to the return message, since the requesting node already has this image.

After the rgb_publisher receives the return message from the convert_rgb_to_gray_scale_server it calls the countPixels server, this is done in the method call_to_count_pixels_server. The action of this server is defined in listing 7. Since leaving the unused feedback field empty resulted in an error we set numberofgrayscalepixels as feedback, we did not actually use this field.

The count_pixels_action_server counts the number of pixels in both the RGB image and the gray scale image. The method count_pixels_in_ros_image, see listing 13, counts the pixels in an image. The action server calls the method on the RGB and the gray scale image before summing them and returning the resulting number of pixels to rgb_publisher. Note that one could also double the number of pixels in either of the images to get the number of pixels in both images. However our implementation is more general, since it also handles two images of different resolutions correctly.

Listing 7: The countPixels action.

```
#goal
sensor_msgs/Image rgb
sensor_msgs/Image grayscale
---
#result
int32 numberofpixels
---
#feedback
int32 numberofgrayscalepixels
```

Listing 8: The Images message.

```
sensor_msgs/Image rgb
sensor_msgs/Image grayscale
int32 numberofpixels
```

The `rgb_publisher` waits until the `count_pixels_action_server` has returned the number of pixels. In the method `talker` it then creates an `Images` message, see listing 8, which contains both images and the total number of pixels in them. This message is broadcast on the topic `/imagesAndPixelsum`.

The node anonymous node, named `listener_13350_1447846059662` in Figure 2, subscribes to the topic `/imagesAndPixelsum`. If it receives a message on this topic it calls the method `show_message`, see listing 14. This method shows both the RGB and gray scale image with the method `show_image` and prints the number of pixels to the log.

It should be noted that we have extracted the functions that convert between CV images and the ROS image format to their own file, see listing 15 to avoid code duplication.

To see the gray scale and RGB images one should execute the commands in listing 9 in different terminals.

Listing 9: The commands that should be executed to run the code for this exercise.

```
# Play the bag file
roslaunch basics exercise3.launch

# Start the node: rgb_publisher
rosrun basiscs rgb_publisher

# Start the service: convert_rgb_to_gray_scale_server
rosrun basiscs rgb_to_gray_scale

# Start the action service: countPixelsServer
rosrun basics count_pixels_action_server

# Start the node that shows the imags:
rosrun basics image_subscriber
```

A Assignment 1

Listing 10: The file assignment.urdf.xacro.

```xml
<?xml version="1.0"?>
<robot name="alice" xmlns:xacro="http://www.ros.org/wiki/xacro">

<property name="M_PI" value="3.1415926535897931"/>
<property name="DEG2RAD" value="0.017453292" />

<gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control
  .so">
        <robotNamespace>/alice</robotNamespace>
    </plugin>
</gazebo>

<!-- top middle Connection Bar from middle connection bar-->
<link name="top_middle_bar">
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <inertial>
        <mass value="4.41176"/>
        <origin xyz="0 0 0"/>
        <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <visual name="visual">
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://alice_description/models/box.
  dae" scale="0.02 0.04 0.3" />
        </geometry>
    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://alice_description/models/box.
  dae" scale="0.02 0.04 0.3" />
        </geometry>
    </collision>
</link>


<joint name="middle_bar_to_top_middle_bar" type="fixed"> <!--
  Connection between Middle bar and bottom rectangular box -->
    <origin rpy="0 0 0" xyz="0 0 ${0.68 / 2 + 0.3 / 2 + 0.02}"/>
    <child link="top_middle_bar"/>
    <parent link="middle_bar"/>
</joint>

<gazebo reference="top_middle_bar">
    <material>Gazebo/Grey</material>
</gazebo>
```

```
<!-- Middle Xtion -->
<link name="middle_xtion_link">
    <inertial>
        <mass value="0.200" />
        <origin xyz="0 0 0" rpy="0 0 0" />
        <inertia ixx="5.8083e-4" ixy="0" ixz="0" iyy="3.0833e-5" iyz=
  "0" izz="5.9083e-4" />
    </inertial>
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <mesh filename="package://alice_description/models/box.
  dae" scale="0.035 0.18 0.025" />
        </geometry>
    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <mesh filename="package://alice_description/models/box.
  dae" scale="0.035 0.18 0.025" />
        </geometry>
    </collision>
</link>


<!-- Gazebo color plugin for xtion -->
<gazebo reference="middle_xtion_link">
    <selfCollide>false</selfCollide>
    <static>true</static>
    <turnGravityOff>false</turnGravityOff>
    <sensor type="depth" name="middle_xtion">
        <pose>0 0 0 0 0 0</pose>
        <always_on>1</always_on>
        <visualize>true</visualize>
        <camera>
            <horizontal_fov>1.047</horizontal_fov>
            <image>
                <width>320</width>
                <height>240</height>
                <format>R8G8B8</format>
            </image>
            <depth_camera></depth_camera>
            <clip>
                <near>0.1</near>
                <far>100</far>
            </clip>
        </camera>
        <plugin name="camera_controller" filename="
  libgazebo_ros_openni_kinect.so">
            <alwaysOn>true</alwaysOn>
            <updateRate>30.0</updateRate>
            <cameraName>front_xtion</cameraName>
            <frameName>front_rgb_optical_link</frameName>
            <imageTopicName>rgb/image_raw</imageTopicName>
            <depthImageTopicName>depth/image_raw</depthImageTopicName
  >
            <pointCloudTopicName>depth/points</pointCloudTopicName>
            <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName
  >
            <depthImageCameraInfoTopicName>depth/camera_info</
  depthImageCameraInfoTopicName>
            <pointCloudCutoff>0.35</pointCloudCutoff>
            <pointCloudCutoffMax>4.5</pointCloudCutoffMax>
            <hackBaseline>0.07</hackBaseline>
            <distortionK1>0.0</distortionK1>
            <distortionK2>0.0</distortionK2>
            <distortionK3>0.0</distortionK3>
            <distortionT1>0.0</distortionT1>
            <distortionT2>0.0</distortionT2>
            <CxPrime>0.0</CxPrime>
            <Cx>0.0</Cx>
            <Cy>0.0</Cy>
            <focalLength>0.0</focalLength>
        </plugin>
    </sensor>
</gazebo>


<joint name="middle_xtion_top_middle_bar" type="fixed"> <!--
  Connection between top bar and front -->
    <origin rpy="0 ${M_PI-M_PI/4} 0" xyz="0.0 0.0 ${0.15+0.0222}"/>
    <child link="middle_xtion_link"/>
    <parent link="top_middle_bar"/>
</joint>



</robot>
```

B Assignment 3

Listing 11: The main node ./src/3/scripts/rgb_publisher.py.

```
#!/usr/bin/env python

import rospy
import actionlib
from sensor_msgs.msg import Image

from basics.srv import *
import basics.msg

def talker(grayscale, rgb, number_of_pixels):
    pub = rospy.Publisher('imagesAndPixelsum', basics.msg.Images,
  queue_size=1)
    msg = basics.msg.Images()
    msg.rgb = rgb
    msg.grayscale = grayscale
    msg.numberofpixels = number_of_pixels
    pub.publish(msg)

def call_to_count_pixels_server(grayscale, rgb):
    client = actionlib.SimpleActionClient('countPixels', basics.msg.
  countPixelsAction)
    client.wait_for_server()
    goal = basics.msg.countPixelsGoal(rgb, grayscale)
    client.send_goal(goal)
    client.wait_for_result()
    return client.get_result().numberofpixels

def call_to_rgb_to_gray_scale_server(image):
    rospy.wait_for_service('convert_rgb_to_gray_scale')
    try:
        convert_rgb_to_gray_scale = rospy.ServiceProxy(
            'convert_rgb_to_gray_scale', RGBtoGrayScale
        )
        resp = convert_rgb_to_gray_scale(image)
        return resp.grayscale
    except rospy.ServiceException, e:
        rospy.logerr("Service call failed: %s"%e)

def subscriber_callback(rgb_image):
    print 'callback'
    grayscale_image = call_to_rgb_to_gray_scale_server(rgb_image)
    number_of_pixels = call_to_count_pixels_server(grayscale_image,
  rgb_image)
    talker(grayscale_image, rgb_image, number_of_pixels)

def listener():
    rospy.Subscriber("/camera/rgb/image_raw", Image,
  subscriber_callback)

if __name__ == "__main__":
    try:
        rospy.init_node(name='rgb_publisher')
    except ROSInitException:
        rospy.logerr("init_node failed; initialization/registration
  failed.")
    except ValueError:
        rospy.logerr("init_node failed; wrong arguments.")
    listener()
    rospy.spin()
```

Listing 12: The service for converting rgb to grayscale ./src/3/scripts/rgb_to_gray_scale.py.

```
#!/usr/bin/env python
import rospy
from basics.srv import *
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from utils import *
```

```python
import cv2
import numpy

import utils

def rgb_to_gray_scale(req):
    bridge = CvBridge()
    cv_image = convert_image_ros_to_cv(bridge, req.rgb)
    cv_gray_scale = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)
    ros_gray_scale_image_msg = convert_image_cv_to_ros(bridge,
   cv_gray_scale, type="8UC1")
    return RGBtoGrayScaleResponse(ros_gray_scale_image_msg)

def convert_rgb_to_gray_scale_server():
    try:
        rospy.init_node('convert_rgb_to_gray_scale_server')
    except rospy.exceptions.ROSInitException:
        rospy.logerr('init_node failed; initialization/registration
   failed.')
    except ValueError:
        rospy.logerr("init_node failed; wrong arguments.")

    rospy.Service(
        name='convert_rgb_to_gray_scale',
        service_class=RGBtoGrayScale,
        handler=rgb_to_gray_scale
    )
    rospy.spin()


if __name__ == "__main__":
    convert_rgb_to_gray_scale_server()
```

Listing 13: The action server that counts the number of pixels `./src/3/scripts/count_pixels_action_server.py`.

```python
#!/usr/bin/env python
import rospy
from cv_bridge import CvBridge, CvBridgeError
import actionlib

import utils
import basics.msg

class CountPixels:

    def count_pixels_in_ros_image(self, ros_image, type):
        try:
            cv_image = utils.convert_image_ros_to_cv(
                self.bridge, ros_image, type)
            height, width = cv_image.shape[:2]
            return height * width
        except CvBridgeError:
            #self.server.set_aborted()
            return -1;

    def __init__(self):
        self.server = actionlib.SimpleActionServer(
            'countPixels',
            basics.msg.countPixelsAction,
            execute_cb=self.execute,
            auto_start = False
        )
        self.bridge = CvBridge()
        self.server.start()

    def execute(self, goal):
        rgb_pixels = self.count_pixels_in_ros_image(goal.rgb, type="
bgr8")

        gray_pixels = self.count_pixels_in_ros_image(goal.grayscale,
type="8UC1")

        pixels = rgb_pixels + gray_pixels
        #rospy.loginfo(pixels)
        result = basics.msg.countPixelsResult()
        result.numberofpixels = pixels
        self.server.set_succeeded(result)

if __name__ == "__main__":
    rospy.init_node(name='countPixelsServer')
    server = CountPixels()
    rospy.spin()
```

Listing 14: The node that subscribes to the end result ./src/3/scripts/image_subscriber.py.

```
#!/usr/bin/env python
import rospy
from cv_bridge import CvBridge, CvBridgeError
import cv2
import utils
from basics.msg import Images

class Image_Subscriber(object):


    def __init__(self):
        self.bridge = CvBridge()

    def spin(self):
        rospy.init_node('listener', anonymous=True)
        rospy.Subscriber("imagesAndPixelsum", Images, self.
show_message)
        rospy.spin()

    def show_image(self, image, window, type='bgr8'):
        cv_image = utils.convert_image_ros_to_cv(
                        self.bridge, image, type)
        cv2.imshow(window, cv_image)


    def show_message(self, msg):
        self.show_image(msg.rgb, 'RGB')
        self.show_image(msg.grayscale, 'grayscale' ,'8UC1')
        rospy.loginfo(msg.numberofpixels)
        cv2.waitKey(1)

if __name__ == "__main__":
    sub = Image_Subscriber()
    sub.spin()
```

Listing 15: The utilities functions `./src/3/scripts/utils.py`.

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import cv2
import numpy

def convert_image_ros_to_cv(bridge, data, type="bgr8"):
    try:
        cv_image = bridge.imgmsg_to_cv2(data, type)
        return cv_image
    except CvBridgeError, error:
        raise error


def convert_image_cv_to_ros(bridge, data, type="bgr8"):
    try:
        ros_image = bridge.cv2_to_imgmsg(data, type)
    except CvBridgeError, error:
        rospy.logerr(error)
    return ros_image
```